1

A DIFFERENCE IN EFFICIENCY BETWEEN SYNCHRONOUS AND ASYNCHRONOUS SYSTEMS*

by

Eshrat Arjomandi
Michael J. Fischer
Nancy A. Lynch

Technical Report #81-03-01

*This material is based upon research supported by the Office of Naval Research under Contracts N00014-80-C-0221 and N00014-79-C-0873, and by the U.S. Army Research Office Contract Number DAAG29-79-C-0155, also NSF #MCS77-15628 and MCS79-24370.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(9) Technical rept.

(14)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
TR-81-03-01

2. GOVT ACCESSION NO.
AD-A097476

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
(6) A DIFFERENCE IN EFFICIENCY BETWEEN SYNCHRONOUS AND ASYNCHRONOUS SYSTEMS

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
(10) Eshrat/Arjomandi, Michael J./Fischer ~~and~~
Nancy A./Lynch

8. CONTRACT OR GRANT NUMBER(s)
(15) [redacted] N00014-80-C-0221, & N00014-79-C-0873; NSF: MCS77-15628 & MCS79-24370; ARO: DAAG29-79-C-0155

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Department of Computer Science, FR-35
University of Washington
Seattle, WA 98195

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
NR 049-456/30 Oct 79 (437)

11. CONTROLLING OFFICE NAME AND ADDRESS
Office of Naval Research, 800 N. Quincy, Arlington, VA 22217, ATTN: Dr. R.B. Grafton / NSF, Washington, D.C. 20550 / U.S. Army Research Office, P.O. Box 12211, Research Triangle Park, NC 27709

12. REPORT DATE
(11) Mar[redacted]81

13. NUMBER OF PAGES
5

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Office of Naval Research
801 N. Quincy
Arlington, VA 22217
ATTN: Dr. R. B. Grafton

(12) 9

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

distributed system, clock, time complexity, synchronization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A system of parallel processes is said to be synchronous if all processes run using the same clock, and it is asynchronous if each process has its own independent clock. For any s, n, a particular distributed problem is defined involving system behavior in n "ports". This problem can be solved in time s by a synchronous system but requires time at least (s-1) log n on any asynchronous system.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

395 224

A DIFFERENCE IN EFFICIENCY BETWEEN SYNCHRONOUS AND ASYNCHRONOUS SYSTEMS*

Eshrat Arjomandi
Dept. of Computer Science
York University
4700 Keele Street
Downsview, Ontario
Canada M3J 1P3

Michael J. Fischer
Dept. of Computer Science
University of Washington
Seattle, Washington 98195

Nancy A. Lynch
Information & Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

ABSTRACT

A system of parallel processes is said to be *synchronous* if all processes run using the same clock, and it is *asynchronous* if each process has its own independent clock. For any s, n, a particular distributed problem is defined involving system behavior at n "ports". This problem can be solved in time s by a synchronous system but requires time at least (s-1) log n on any asynchronous system.

INTRODUCTION

A system of parallel processes is said to be *synchronous* if all processes run using the same clock, so the processes operate in lock-step, and it is *asynchronous* if each process has its own independent clock. Examples of synchronous systems are large centralized multiprocessing computers and VLSI chips containing many separate parallel processing elements. Examples of asynchronous systems are distributed computer networks and I/O systems for conventional computers.

In this paper, we compare time efficiency of a simple model of a synchronous system with a similar asynchronous model. We bound the number of processes that can access any particular communication channel, and that restriction is crucial to our results. For s, n $\in$ N, we define a particular distributed problem involving n "ports". It can be solved in time s on a synchronous system but we show it requires time at least $(s-1) \lfloor \log_b n \rfloor$ on any asynchronous system. Here b is a constant reflecting the communication bound in the model, whose precise definition is given in the next section. If we strengthen the communication system slightly to permit a single designated process to broadcast to all the others, or if we provide each process with access to a global clock, then the asynchronous model can solve the problem in time O(s).

THE MODELS

We use a version of the model of a concurrent system defined in [LF79,LF81]. Briefly, it consists of collections P of *processes* and X of *shared variables*. The *global state* of the system consists of the internal state of each process together with the value of each shared variable. A *step* is an atomic action which consists of simultaneous changes to the state of some process and the value of some shared variable. Formally, a *step* σ is a pair of triples ((s,p,t),(u,x,v)) where s, t are possible internal states of process p, and u, v are possible values of variable x. We define *process*(σ) = p and *variable*(σ) = x and say σ *involves* p and *accesses* x. Step σ is *applicable* to any global state in which process p has internal state s and variable x contains value u. The effect of performing σ is to change the state of p to t and simultaneously to change the value of x to v.

A system is specified by describing P, X, an initial global state, and a set OKSTEPS of possible steps. A process p *blocks* in a global state g if there is no step σ in OKSTEPS applicable to g with process(σ) = p. In this paper, we require our systems to be non-blocking for all processes and all global states.

Let x $\in$ X and define *locality*(x) = {process(σ) : $\sigma \in$ OKSTEPS and variable(σ) = x}. A system is *b-bounded* if |locality(x)| $\le$ b for every x $\in$ X.

A *computation* of a system is a finite or infinite sequence of steps in OKSTEPS such that the first step is applicable to the initial global state, and each succeeding step is applicable to the state resulting from the application of the previous step. The *result* of a finite computation is the global state after applying the sequence. An infinite computation is *admissible* if every process appears in infinitely many steps of the

*This material is based upon research supported by the Office of Naval Research under Contracts N00014-80-C-0221 and N00014-79-C-0873, and by the U.S. Army Research Office Contract Number DAAG29-79-C-0155, also NSF #MCS77-15628 and MCS7924370.

sequence.

A *round* is any sequence of steps such that every process appears at least once in the sequence. A *minimal round* is a round such that no proper prefix is a round. Every sequence of steps can be uniquely partitioned into *segments* such that every segment is a round, except possibly for the last if the sequence is finite, and every round is minimal. We call this a partition into minimal rounds, even though the last segment is not necessarily a round.

A sequence of steps is *synchronous* if in the unique partition into minimal rounds:

(1) No two steps in the same round involve the same process;

(2) No two steps in the same round access the same variable.

(1) and (2) together imply that the steps in each round are independent and can be performed in any order, or simultaneously, with the same result.

The *run time* for a finite sequence of steps is defined to be the number of segments in the partition into minimal rounds. (This definition is equivalent to the one in [LF81] which says it is the longest amount of elapsed real time that the system could take to execute the sequence, subject to the constraint that the time delay between two steps of the same process is at most unity. For synchronous systems, this definition is also equivalent to the more usual one which simply counts the number of synchronous steps of the system, where one synchronous step consists of the simultaneous execution of a step by each process.)

Finally, a *synchronous system* is a concurrent system whose allowable computations are all of its infinite synchronous computations. An *asynchronous system* is a concurrent system whose allowable computations are all of its infinite admissible computations.

THE PROBLEM

We now define a particular behavior for a concurrent system. Let $Y \subseteq X$ be a distinguished set of variables called *ports*. A *port event* is any step that accesses a port. A *session* is any sequence of steps containing at least one port event for every port. A computation *performs* s sessions if it can be partitioned into s segments, each of which is a session. An infinite computation is *ultimately quiescent* if it contains only a finite number of port events. The *time to quiescence* of an ultimately quiescent sequence is the run time of the shortest prefix containing all port events.

Let $s, n \in \mathbb{N}$. The (s,n)-*session problem* is the problem of finding a concurrent system with n ports such that every allowable computation performs (at least) s sessions and is ultimately quiescent.

Note that the (s,n)-session problem, like the mutual exclusion and dining philosophers problems, concerns possible orderings of sequences of events rather than the computation of particular outputs. It is an abstraction of the synchronization needed in many natural problems. Consider, for example, a simple message distribution system in which a sending process writes a sequence of s messages one at a time on a board visible to all and waits after each message until all n other processes have read the message. Whatever protocol insures that the sender has waited sufficiently long will also solve the (s,n)-session problem.

MAIN RESULT

We show that any asynchronous b-bounded system solving the (s,n)-session problem requires time at least $(s-1) \lfloor \log_b n \rfloor$ to quiescence, whereas there is a trivial synchronous system which solves the problem in time exactly s. This is the first example we know of, of a problem for which an asynchronous system is provably slower than a synchronous one, and it shows that a straightforward step-by-step and process-by-process simulation of an n-process synchronous system by an n-process asynchronous one *necessarily* loses a factor of $\log_b n$ in speed.

The result is even more surprising when one realizes that the trivial asynchronous system with one process per port (and no communication among the processes) in which each process does nothing except access a port on each step in fact performs s sessions within time s. The difficulty is that no process knows when time s has elapsed (due to the lack of a global system "clock"), nor does it know when the s sessions have in fact been achieved, so none of the processes knows when to stop accessing its port.

A procedure which does work is for a process associated with each port to perform a port event, broadcast that fact and then wait until it has heard that all other port processes have performed their port events and that the session has been completed. This is repeated s times. By making the port processes the leaves of a tree network, the necessary communication for one session can be accomplished in time O(log n); hence, the total time to quiescence for the solution is O(s log n). It seems very inefficient to wait after each port event, and one might try to invent clever schemes to increase the concurrency in the system. Our lower bound shows, however, that this method is optimal to within a constant factor, so only a limited amount of improvement is possible.

We now present the formal results.

Theorem 1. For all $s, n \in \mathbb{N}$, there is a 1-bounded synchronous system which solves the (s,n)-session problem, such that the time to quiescence for each allowable computation is s.

Proof. The system has n processes, one corresponding to each port. Each process accesses its port on each of its first s steps and then ceases performing port events. In every infinite synchronous computation, each of the first s minimal rounds constitutes a session, and the system becomes quiescent after s rounds. Hence, the system solves the (s,n)-session problem in time s.

□

Theorem 2 (Main Result). Assume $b, s, n \in \mathbb{N}$, $b \geq 2$. For every b-bounded asynchronous system which solves the (s,n)-session problem, the time to quiescence is at least $(s-1) \lfloor \log_b n \rfloor$ for some allowable computation.

The proof of Theorem 2 involves a series of three lemmas about a particular partial ordering of steps of a computation. (The ordering represents a kind of logical dependency.) We break up the proof of Theorem 2 by presenting the lemmas before the main proof. These lemmas and their proofs are self-contained and depend only on the properties given below. For better intuitive motivation, however, the reader may wish to read the main proof before reading the lemmas.

Let R be the set $\{1,\ldots,a\}$ (of "round" numbers), P a finite set (of "processes"), X a set (of "variables"). Let D be a set having mappings round : $D \rightarrow R$, proc : $D \rightarrow P$ and var : $D \rightarrow X$. Assume that for every pair $(r,p) \in R \times P$, there is exactly one $\sigma \in D$ having $\mathrm{round}(\sigma) = r$ and $\mathrm{proc}(\sigma) = p$. Let $\mathrm{loc}(x) = \{\mathrm{proc}(\sigma) : \sigma \in D \text{ and } \mathrm{var}(\sigma) = x\}$. Let $b \geq 2$ and assume $|\mathrm{loc}(x)| \leq b$ for all $x \in X$.

Let $\leq$ be a partial order on D, and write $\sigma \underset{1}{\leq} \tau$ to indicate that $\sigma < \tau$ and there is no ρ with $\sigma < \rho < \tau$. Assume that $\leq$ has the following properties:

(i) If $\sigma \underset{1}{\leq} \tau$, then either $\mathrm{var}(\sigma) = \mathrm{var}(\tau)$ or $\mathrm{proc}(\sigma) = \mathrm{proc}(\tau)$.

(ii) If either $\mathrm{var}(\sigma) = \mathrm{var}(\tau)$ or $\mathrm{proc}(\sigma) = \mathrm{proc}(\tau)$, then σ and τ are $\leq$-comparable.

(iii) If $\sigma \leq \tau$, then $\mathrm{round}(\sigma) \leq \mathrm{round}(\tau)$.

Finally, let $\mathrm{dep}(\sigma) = \{\mathrm{var}(\tau) : \sigma \leq \tau\}$.

Lemma 1 (Monotonicity). If $\sigma_1 \leq \sigma_2$, then $\mathrm{dep}(\sigma_2) \subseteq \mathrm{dep}(\sigma_1)$.

Proof. Obvious from the definition of dep.

□

Lemma 2. Let $\sigma \in D$, $\mathrm{round}(\sigma) = r$, $\mathrm{var}(\sigma) = x$. Let $C = \{\tau \in D : \mathrm{round}(\tau) = r + 1 \text{ and } \mathrm{proc}(\tau) \in \mathrm{loc}(x)\}$.

Then $\mathrm{dep}(\sigma) \subseteq \bigcup_{\tau \in C} \mathrm{dep}(\tau) \cup \{x\}$.

Proof. Proof is by induction on $\leq$, beginning with maximal elements. Let $\sigma \in D$ and assume the lemma holds for all $\tau > \sigma$. Assume r, x and C are defined from σ as in the statement of the lemma. If there exists $\sigma' \in D$ with $\mathrm{var}(\sigma') = x$ and $\sigma' > \sigma$, then fix σ' as the smallest such member of D. (Property (ii) insures that σ', if it exists, is defined uniquely.) Similarly, if there exists $\sigma'' \in D$ with $\mathrm{proc}(\sigma'') = \mathrm{proc}(\sigma)$ and $\sigma'' > \sigma$, then fix σ'' as the smallest such member of D. Define $B' = \mathrm{dep}(\sigma')$ if σ' exists, ϕ otherwise, and $B'' = \mathrm{dep}(\sigma'')$ if σ'' exists, ϕ otherwise. Then properties (i) and (ii) and monotonicity show that $\mathrm{dep}(\sigma) \subseteq B' \cup B'' \cup \{x\}$. It suffices to show that $B' \cup B'' \subseteq \bigcup_{\tau \in C} \mathrm{dep}(\tau) \cup \{x\}$.

We first consider B', and assume σ' exists. (If σ' does not exist, there is nothing to prove.) By induction, $B' \subseteq \bigcup_{\tau \in C'} \mathrm{dep}(\tau) \cup \{x\}$, where $C' = \{\tau \in D : \mathrm{round}(\tau) = \mathrm{round}(\sigma') + 1 \text{ and } \mathrm{proc}(\tau) \in \mathrm{loc}(x)\}$. For every $\tau' \in C'$, there exists $\tau \in C$ with $\mathrm{proc}(\tau) = \mathrm{proc}(\tau')$. Property (ii) shows that τ and τ' are $\leq$-comparable; property (iii) shows that $\tau \leq \tau'$. Monotonicity implies that $\mathrm{dep}(\tau') \subseteq \mathrm{dep}(\tau)$. Thus,

$B' \subseteq \bigcup_{\tau \in C} \mathrm{dep}(\tau) \cup \{x\}$, as needed.

Finally, we consider B'', and assume σ'' exists. Then the properties of D and $\leq$ show that $\mathrm{round}(\sigma'') = r + 1$, so that $\sigma'' \in C$.

Thus, $B'' \subseteq \bigcup_{\tau \in C} \mathrm{dep}(\tau)$, as needed.

□

Lemma 3. For each $\sigma \in D$, it is the case that

$$|\mathrm{dep}(\sigma)| \leq \frac{b^{a-\mathrm{round}(\sigma)+1}-1}{b-1}.$$

Proof. We proceed by induction on $k = \mathrm{round}(\sigma)$. starting with $k = a$ and working backwards.

BASIS: $k = a$.. By Lemma 2, $\mathrm{dep}(\sigma) \subseteq \{\mathrm{var}(\sigma)\}$, so $|\mathrm{dep}(\sigma)| \leq 1$, as needed.

INDUCTION: $1 \leq k < a$. By Lemma 2, we have $|\mathrm{dep}(\sigma)| \leq \sum_{\tau \in C} |\mathrm{dep}(\tau)| + 1$, where C is defined as in Lemma 2. Each $\tau \in C$ has $\mathrm{round}(\tau) = k + 1$, so by induction, $|\mathrm{dep}(\tau)| \leq \frac{b^{a-k}-1}{b-1}$. Also, $|C| \leq b$. Hence, $|\mathrm{dep}(\sigma)| \leq$

$$b \cdot \left[\frac{b^{a-k}-1}{b-1}\right] + 1 = \frac{b^{a-k+1}-1}{b-1}, \text{ as needed.}$$

□

Proof of Theorem 2. Assume an asynchronous system which solves the (s,n)-session problem. Enumerate the processes arbitrarily. Construct an infinite admissible computation α by running the processes in round-robin order (one step of process 1, one of process 2,..., one step of process q, one step of process 1,...). Each round-robin round is minimal and contains exactly one step of each process; hence, the time to perform the first r rounds is exactly r. Because we assume a correct solution, this computation is ultimately quiescent. Let t be the time to quiescence for this computation. Then round t is the last round at which any port event occurs. We wish to show $t \ge (s-1)\lfloor \log_b n \rfloor$.

Let $\alpha = \beta\gamma$, where β contains the first t rounds of α, and γ is the remaining tail. Our strategy is to construct a new infinite admissible computation $\alpha' = \beta'\gamma$, where β' is a reordering of the steps of β that results in the same global state as β, but β' performs at most $t/\lfloor \log_b n \rfloor + 1$ sessions. Since no port events occur in γ, it follows that α' performs at most $t/\lfloor \log_b n \rfloor + 1$ sessions. Since α' is an infinite admissible computation for the system, $t/\lfloor \log_b n \rfloor + 1 \ge s$ and our result follows.

To construct β', we first construct a partial order of the steps in β, representing "dependency". (Formally, the domain of the partial order consists of ordered pairs (i,ξ_i), where ξ_i is the ith step of β.) For every pair of steps σ, τ in β, we let $\sigma \underset{\beta}{\le} \tau$ if σ precedes τ in β and either process(σ) = process(τ) or variable(σ) = variable(τ). Close $\underset{\beta}{\le}$ under transitivity. $\underset{\beta}{\le}$ is a partial order, and every total order of the steps of β consistent with $\underset{\beta}{\le}$ is a computation which leaves the system in the same global state as β. (Clearly β itself defines such a total ordering.)

Now, let $m = \lceil t/\lfloor \log_b n \rfloor \rceil$, and write $\beta = \beta_1 \ldots \beta_m$, where β_k consists of $\lfloor \log_b n \rfloor$ minimal rounds, $1 \le k < m$. Let y_0 be an arbitrary port. For $k = 1,\ldots,m$, we define inductively a port y_k and two sequences of steps ϕ_k and ψ_k, as follows. There are two cases. First, if there is some port which is not accessed by any step of β_k, then take y_k to be that port and let $\phi_k = \Lambda$ (the null sequence) and $\psi_k = \beta_k$. Otherwise, let τ_k be the first step in β_k which accesses y_{k-1}. We now wish to apply Lemma 3, to the subordering of $\underset{\beta}{\le}$ defined by restriction to rounds with numbers $(k-1)\lfloor \log_b n \rfloor+1,\ldots, k\lfloor \log_b n \rfloor$ inclusive. The mapping "rounds" required for the lemmas is obtained by renumbering the rounds in the same order. Mappings "proc" and "var" are obtained from the mappings "process" and "variable" respectively. It is straightforward to see that the necessary properties of D and $\le$ are satisfied. Then by Lemma 3, we see that $|\mathrm{dep}(\tau_k)| \le$

$\dfrac{b^{\lfloor \log_b n \rfloor-1+1}-1}{b-1} \le n-1$. Since there are n ports, this means that there must exist a port y_k and a step σ_k such that

(i) σ_k is the last step in β_k which accesses y_k;

(ii) it is false that $\tau_k \underset{\beta}{\le} \sigma_k$.

Thus, adding the relation $\sigma_k \underset{\beta}{\le} \tau_k$ to $\underset{\beta}{\le}$ and closing under transitivity results in another partial order $\underset{k}{\le}$. Choose any total ordering of the steps in β_k consistent with $\underset{k}{\le}$. Let ϕ_k be the longest prefix of that ordering not containing any step accessing y_{k-1}, and let ψ_k be the remainder. This is illustrated in Figure 1.

In either case, ϕ_k does not contain any step which accesses y_{k-1} and ψ_k does not contain any step which accesses y_k.

Let $\beta' = \phi_1\psi_1\phi_2\psi_2\cdots\phi_m\psi_m$. β' is consistent with $\underset{\beta}{\le}$, but β' contains at most $m \le t/\lfloor \log_b n \rfloor + 1$ sessions, since each session must contain steps on both sides of some ϕ_k-ψ_k boundary. (If a sequence of steps were completely contained in $\psi_{k-1}\phi_k$, for example, then it would fail to contain a step accessing port y_{k-1}.)
$\square$

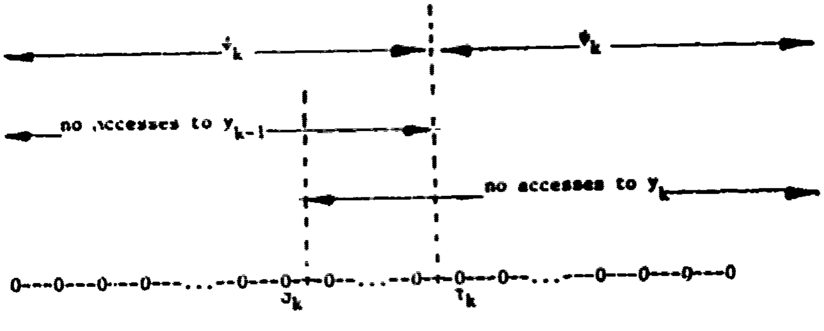

Figure 1. A total ordering of steps in β_k consistent with $\underset{k}{\le}$.

RESULTS FOR MORE GENERAL MODELS

If the model is generalized by removing the bound on the number of processes which can access a shared variable, then a single communication variable shared by n port processes can be used to construct an easy O(s) solution.

In fact, if the original model is only generalized slightly by allowing one of the shared variables to be read by an arbitrary number of processes (but only to be changed by one process), then an O(s + log n) solution is possible. In more detail, we use a shared variable, the message board, which every process can read but only one fixed process, the supervisor, can change. Each port has a corresponding port process, and there are additional communication processes whose job it is to pass messages through a tree network from the port processes back to the supervisor. The message board contains an integer which we call a "clock" value. The supervisor alternately increments the clock and reads the messages being sent back. Each port process repeatedly performs a cycle of reading the clock, performing a port event, and sending a message back to the supervisor, through the network, which contains the clock value just read and the port identifier.

If c_1 and c_2 are two successive clock values sent by port process i, then a port event must occur at port i sometime after the clock assumes value c_1 and before the clock assumes value $c_2 + 1$. By naturally combining this information about all ports, the supervisor can construct a sequence $0 = b_0 < b_1 < b_2 < \ldots < b_s$ such that for each j, a session is guaranteed to occur between the times when the clock first assumes values b_j and b_{j+1}. (Specifically, let $c_{i1}, c_{i2}, \ldots$ denote denote the successive clock values sent by port process i, $1 \le i \le n$. Then define

$b_j = \max \{c_{i(k+1)} + 1 : 1 \le i \le n$ and k is the smallest index such that $c_{ik} \ge b_{j-1}\}$, for each j, $1 \le j \le s$.) After the supervisor constructs this entire sequence, it knows that at least s sessions have in fact occurred, at which time it puts a "STOP" message on its message board. When the port processes read the "STOP" message, they stop performing port events.

It is easy to see that this construction solves the (s,n)-session problem. We argue that it satisfies the required O(s + log n) time bound.

First, we consider message transmission time. Since we are not assuming any upper bound on size of variables, the tree network can guarantee (by concatenating messages) that any message can be sent as soon as a process is ready to send it, and also that any message sent by time t is received by the supervisor by time t + O(log n).

Next, we claim that for each j, $1 \le j \le s$, the elapsed time between when the clock first assumes values b_{j-1} and b_j is bounded above by a constant. For, from the time when the clock first assumes value b_{j-1}, it is at most a fixed constant amount of time before all port processes have read the clock, performed port events, sent messages containing clock values $\ge b_{j-1}$, and read the clock once again. Thereafter, it is at most one time unit before the clock is incremented again, thereby assuming value b_j.

Thus, the total elapsed time until the clock assumes value b_s is O(s). Thereafter, within time O(log n), the supervisor has received all the needed messages and can deduce that s sessions have occurred and display the "STOP" message. Three time units later, all port processes will have read the "STOP" message and will have stopped performing port events.

REFERENCES

[LF79] N.A. Lynch and M.J. Fischer. On Describing the Behavior and Implementation of Distributed Systems. In Semantics of Concurrent Computation, ed. G. Kahn, Vol. 70 of Lecture Notes in Computer Science series, Springer-Verlag, 1979, 147-171.

[LF81] N.A. Lynch and M.J. Fischer. On Describing the Behavior and Implementation of Distributed Systems. Theoretical Computer Science, 13, North-Holland Publishing Company, 1981, 17-43.

DISTRIBUTION LIST

Office of Naval Research Contract N00014-80-C-0221
Michael J. Fischer, Principal Investigator

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
(1 copy)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Dr. R. B. Grafton, Scientific Officer (1 copy)
Information Systems Program (437) (2 copies)
Code 200 (1 copy)
Code 455 (1 copy)
Code 458 (1 copy)

Office of Naval Research
Branch Office, Pasadena
1030 East Green Street
Pasadena, CA 91106

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375
(6 copies)

Office of Naval Research
Resident Representative
University of Washington, JD-27
422 University District Building
1107 NE 45th Street
(1 copy)

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380
(1 copy)

Naval Ocean Systems Center
Advanced Software Technology Division
Code 5200
San Diego, CA 92152
(1 copy)

Mr. E. H. Gleissner
Naval Ship Research and Development Center
Computation and Mathematics Department
Bethesda, MD 20084
(1 copy)

Captain Grace M. Hooper (008)
Naval Data Automation Command
Washington Navy Yard
Building 166
Washington, D.C. 20374
(1 copy)

Defense Advanced Research Projects Agency
ATTN: Program Management/MIS
1400 Wilson Boulevard
Arlington, VA 22209
(3 copies)

Professor Eshrat Arjomandi
Department of Computer Science
York University
4700 Keele Street
Downsview, Ontario
Canada M3J 1P3
(1 copy)

Professor Nancy A. Lynch
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
(1 copy)

Office of Naval Research
Branch Office, Chicago
536 South Clark Street
Chicago, IL 60605
(1 copy)

Dr. John Cherniavsky
Program Director
Theoretical Computer Science
National Science Foundation
Washington, D.C. 20550
(2 copies)

Department of the Army
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709
(1 copy)